2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Classifications of computing sites to handle numerical variability

Tristan Glatard*† and Alan C. Evans*
*McConnell Brain Imaging Centre, Montreal Neurological Institute, McGill University, Canada.
†University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France.
Email: {first.last}@mcgill.ca

***Abstract*—Programs may generate different results depending on the computing system where they are executed. Consequently, distributed applications are restricted to a few computing sites to avoid being flawed by numerical variations. We propose algorithms to increase the use of infrastructures while controlling the risk created by numerical variations. Our algorithms rely on a classification of the computing sites updated regularly depending on (i) the amount of observed numerical differences, and (ii) the number of tasks to execute. We simulate our algorithms on a 40-site infrastructure using SimGrid, in 3 different configurations. Results show that for one of the configurations, our algorithm speeds up the execution by a factor of about 4 in 10% of the cases. For the remaining cases and the other 2 configurations, we show that sites cannot be aggregated unless a high risk of numerical variations is tolerated. We conclude that site classifications are a promising approach to handle numerical variability among computing sites. Results could be further improved by integrating more *a-priori* information in the classifications.**

I. Introduction

Programs may generate different results depending on the computing system where they are executed [1]. We call such situations *numerical accidents*. Even when homogeneous hardware is used, software heterogeneity may impact results due to the evolution of library implementations, in particular mathematical libraries. In brain image analysis, a field of particular interest for us, numerical accidents have a measurable impact on the performance of applications, and they might even produce an effect comparable to the one measured in some studies [2]. Numerical accidents hamper the replication of analyses, therefore reproducible science.

To avoid numerical accidents, applications executed on distributed computing systems must be restricted to subsets of sites that are consistent numerically, i.e. where results are replicable. Hence, the following configurations may be studied:

- **C1**: compute all possible results;
- **C2**: compute one particular result;
- **C3**: compute any result.

For instance, **C1** may be used to evaluate the reproducibility of an application on the infrastructure, **C2** to replicate a study previously executed, and **C3** to produce a result as quickly as possible.

The state-of-the-art solution to address numerical accidents on grid infrastructures is to limit applications to a single computing site, as done in platforms such as CBRAIN [3]. This approach under-exploits the infrastructure and is limited to configuration **C2**. Other platforms, for instance the Virtual Imaging Platform [4] ignore the issue, which might lead to incorrect results. We propose algorithms to increase the use of computing sites in the 3 configurations, while controlling the risk of numerical accidents. Our algorithms classify sites based on a risk model derived exclusively from observations. We assume that computing sites are equipped with schedulers (e.g. batch schedulers), and that the infrastructure has metaschedulers to dispatch tasks to the sites. Discussing the algorithms implemented by such schedulers is out of our scope.

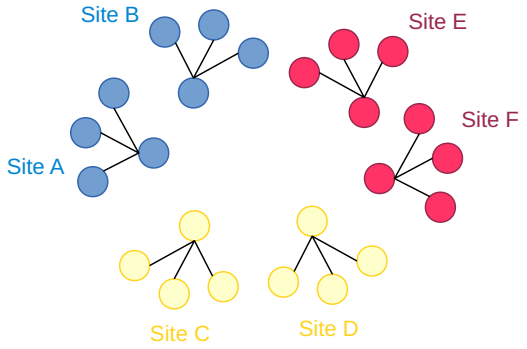


Fig. 1: A classification of computing sites showing 6 sites and 3 classes.

We introduce our algorithms in Section II, we describe their simulation on a 40-site grid in Section III, and we evaluate them in Section IV.

II. Algorithms

As illustrated on Figure 1, our algorithms rely on a classification of computing sites where all the sites in a class supposedly produce identical results. Finding such a classification is not straightforward due to the following issues:

- The number of accidents observed between two sites does not define a distance in the set of computing sites because it does not verify the triangle inequality when different task sets are executed on different site pairs. For instance, given three sites x, y, and z, we may observe some accidents between site x and site z, but 0 accidents between site x and site y as well as between site y and site z.
- We cannot assume that a classification is valid permanently. Even when hardware and software configurations do not change, there is always a risk that accidents happen during the processing of a specific data set, which needs to be modeled.

978-1-4799-8006-2/15 $31.00 © 2015 IEEE
DOI 10.1109/CCGrid.2015.167

To address these issues, we define a "compatibility" relation between two sites depending on (i) the amount of previously observed accidents, and (ii) the number of tasks to execute. Compatible sites may become incompatible when new accidents are observed, or when a too large number of tasks need to be executed. Sites are finally classified based on this relation.

A. Notations

- Sets
  - $\#X$: number of elements in X.
  - []: empty set.
- Computing sites
  - S: set of all computing sites on the infrastructure.
  - S_t: set of sites where task t was executed.
- Tasks
  - T_x: set of tasks in T executed on site x, i.e. $\{t \in T, x \in S_t\}$.
  - t_x: result produced by task t executed on site x.
  - $A_{x,y}$: numerical accidents between site x and site y, i.e. $\{t \in T_x \cap T_y, t_x \neq t_y\}$
- Variable types
  - `int`: integer.
  - `prob`: probability, i.e. real number in [0,1].
  - `task`: task executed on the infrastructure.
  - `site`: computing site.
  - `class`: set of sites.
  - `[type]`: set of elements of type `type`.

B. Site classification

We rely on the theory described in Chapter 14 of [5] for the estimation of extreme phenomena, which states that the probability to observe 0 accidents among N' future observations given that n accidents happened among N previous observations is:

$$p(N', n, N) = \frac{N+1}{N+N'+1} \frac{\binom{N}{n}}{\binom{N+N'}{n}} \quad (1)$$

This formula is based on the following summation on the risk factor to have numerical accidents:

$$p(N', n, N) = \int_0^1 (1-\lambda)^{N'} f_{n,N}(\lambda) d\lambda$$

$f_{n,N}$ is the density of probability of the risk factor given that n accidents happened among N observations:

$$f_{n,N}(\lambda) = \frac{\lambda^n (1-\lambda)^{N-n}}{\int_0^1 x^n (1-x)^{N-n} dx}$$

Based on Equation 1, we define a compatibility relation between two sites x and y:

$$\text{COMPATIBLE}(x, y) \Leftrightarrow p(N', \#A_{x,y}, \#T_x \cap T_y) \geq \alpha \quad (2)$$

Sites are COMPATIBLE if and only if the probability to execute N' tasks on these sites without encountering any numerical accident is greater than α.

Algorithm 1 describes the classification of a set S of sites where a task set T was previously executed. Each task in T may have been executed on one or several sites in S. N'' denotes the number of future tasks to execute. α is the lower bound used in Equation 2. Procedure CLASSIFY: starting from an empty classification, sites are iteratively assigned to classes. Procedure ASSIGN: if a class exists where all the sites are compatible with the site to assign, then the site is assigned to this class; otherwise, the site is assigned to a new class. Procedure COMPATIBLE_CLASS: site compatibility is evaluated based on relation COMPATIBLE; the number of tasks to execute based on the existing observations (N') is the sum of the number of future tasks to execute (N''), and the number of previous tasks that were executed exclusively on site x or on site y ($\#T_x + \#T_y - 2\#T_x \cap T_y$).

Algorithm 1 Site classification

```
procedure CLASSIFY([site] S, [task] T, prob α, int N")
    [class] C = []
    for s ∈ S do
        ASSIGN(s,C,T,α,N")
    return C
procedure ASSIGN(site x, [class] C, [task] T, prob α, int N")
    for c ∈ C do
        if COMPATIBLE_CLASS(x,c,T,α,N") then
            c = c ∪ {x}                    ▷ Adds site x to class c
            return
    C = C ∪ {{x}}                          ▷ Creates a new class
procedure COMPATIBLE_CLASS(site x, class c, [task] T, prob α, int N")
    for y ∈ C do
        N'=N"+#T_x+#T_y-2#T_x ∩ T_y
        if p(N',#A_{x,y},#T_x ∩ T_y) <α then return false
    return true
```

C. C1: Computing all results

Our algorithm for **C1** is a straightforward application of site classification:

1) Classify sites using Algorithm 1.
2) Execute the workload in all classes.

This algorithm executes the workload once for each *class*. Instead, the state-of-the-art solution executes the workload once for each *site*.

D. C2: Computing one result

Assuming that the wanted result is the one computed by site x, then our algorithm is as follows:

1) Classify sites using Algorithm 1.
2) Execute the workload in the class containing site x.

This algorithm uses all the sites in the class containing site x. Instead, the state-of-the-art solution uses only site x.

E. C3: Computing any result

Assuming that an oracle exists to order sites and to order classes by decreasing value of performance, then our algorithm is as follows:

	min	max	average
Hosts per cluster	6	93	38.2
Host power (Gflops)	4.7	30.1	12.5

TABLE I: Hosts of the simulated platform.

	Bandwidth (Gb/s)	Latency (ms)
Within cluster	1	0.1
Between clusters	10	0.1

TABLE II: Network of the simulated platform.

1) Classify sites using Algorithm 1.
2) Execute the workload in the first class in the order returned by the oracle.

This algorithm uses the best-performing *class* according to the oracle. Instead, the state-of-the-art solution uses the best-performing *site* according to the oracle.

III. Simulation

We simulated our algorithms on a grid of computing sites using the SimGrid simulator [6]. SimGrid simulations consist of (i) a platform description comprising hosts, clusters, routers, network links and routes, (ii) a deployment of services on hosts of the platform, (iii) a workload executed by the services.

a) Platform: We used the description of the Grid'5000 infrastructure [7] as available in SimGrid version 3.11. This platform has 40 clusters located in 9 different French cities, each with a different network domain. Host performance is homogeneous within a cluster but it varies across clusters, as shown in Table I. All local networks within clusters have identical characteristics, and all network links between clusters are identical too, as shown on Table II.

b) Deployment: We deployed batch systems and nodes to dispatch tasks to hosts of a cluster, and meta-schedulers to dispatch tasks to clusters of a class. Batch systems, nodes and meta-schedulers were taken from the Simbatch [8] extension of SimGrid. To take hardware heterogeneity into account, we modified Simbatch so that it accepts tasks defined with a number of operations rather than with a duration. Batch systems were configured to implement conservative backfilling.

c) Workload: The workload consisted of the iterative submission of batches of 150 tasks simulating brain image processing as implemented in Freesurfer [9]. Task cost was set to 1117958.9 Gflops so that tasks lasted 24 hours in average on the simulated platform, which is a common execution time for Freesurfer. Tasks requested a walltime of 72 hours to avoid being killed by the batch systems. Task input and output files were 25 MB and 450 MB, respectively.

d) Accidents: Numerical accidents were simulated only between sites belonging to different cities, with a probability ρ. To do that, tasks were assigned a return code of 1 with probability ρ', and a return code of 0 with a probability 1-ρ'. We used $\rho' = -\frac{\sqrt{1-2\rho}-1}{2}$ so that the probability that two tasks have different return codes, i.e. $p = 2\rho'(1-\rho')$, equals ρ. Return codes were ignored for tasks executed on different sites of the same city.

Parameter	Values
#S	{2, 5, 10, 20, 40}
α	{0.1, 0.3, 0.5, 0.7, 0.9}
ρ	$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1/2\}$

TABLE III: Experiment parameters

e) C2 and C3: In configuration **C2**, the wanted site x was selected as the first one in the alphabetical order. In configuration **C3**, the oracle sorted sites and classes by decreasing number of total power (i.e. number of nodes multiplied by power of nodes).

IV. Experiment

A. Setup

We simulated our algorithms using different numbers of sites, different values of α, and different values of ρ, as reported in Table III. Sites were randomly picked on the platform. For each site number, 3 repetitions of the experiment were done, corresponding to 3 different site sets. For each value of α, each value of ρ, and each configuration, the workload was iterated once with the state-of-the-art algorithm (all subsequent iterations would give the same result), and 100 times with our algorithm (an iteration uses the site classification obtained from the previous iteration). An iteration of the state-of-the-art simulation was implemented as follows:

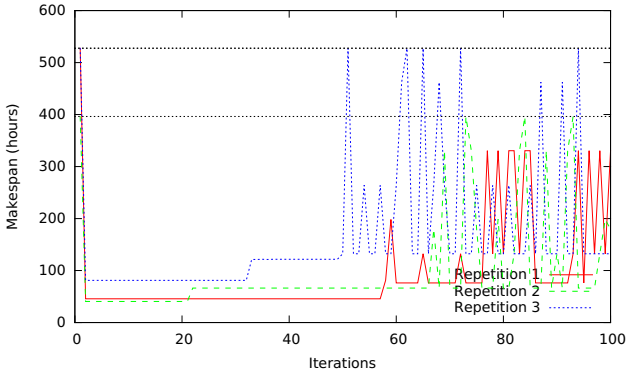
- Generate 150 Freesurfer tasks;
- Submit 150 tasks to each batch system (**C1**), or to the batch system containing a particular site (**C2**), or to the batch system ranked first by the oracle (**C3**).

Iterations of our algorithm consisted of the following steps:

- Generate 150 Freesurfer tasks;
- Classify sites based on the task return codes produced by previous iterations;
- Deploy a metascheduler for each class;
- Submit 150 tasks to all metaschedulers (**C1**), or to the metascheduler managing a particular site (**C2**), or to the metascheduler ranked first by the oracle (**C3**).

B. Results

1) Effect of α: Figure 2 displays the simulated makespan of our workload executed in configuration **C1** on 5 sites, for $\rho = 0.5$ and increasing values of α. The makespan is measured as the duration between the submission time of the first task, and the completion time of the last task. For each repetition, the makespan obtained with the state-of-the-art algorithm is represented with dotted black lines. The makespan obtained with our algorithm oscillates between a maximal value, where all sites are in a separate class, and a minimal value, where some classes have more than 1 site. The maximal value also corresponds to the makespan obtained with the state-of-the-art solution. Oscillations are a direct implication of our design assumption that a classification is not permanently valid: when the number of tasks executed in a class increases, then N' increases in equation 2, which lowers p. When p is lower than α, classes have to be split to increase the number of observations ($\#T_1 \cap T_2$) and bring p above α again.



(a) $\alpha = 0$ 1.

Parameter α mainly has two effects:

1) It reduces the frequency of low makespan values.
2) It regularizes the evolution of the makespan. When α is above 0.5, the makespan becomes pseudo-periodical, with a pseudo-period increasing with α.

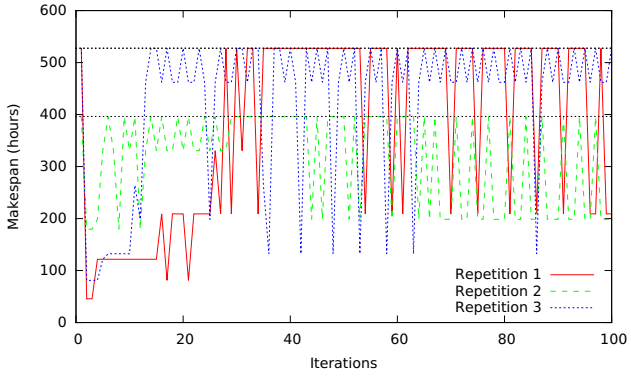
The first effect is a consequence of the definition of our classification: sites becomes less COMPATIBLE when α increases. The second effect comes from the fact that the classification is less sensitive to the occurrence of accidents when α increases.

The speed-up factor obtained with our algorithm compared to the state of the art depends on the average number of sites per class. On Figure 2, it varies between 1 (no speed-up) and 11.5. For $\alpha = 0.9$, about 10% of the executions are significantly accelerated.

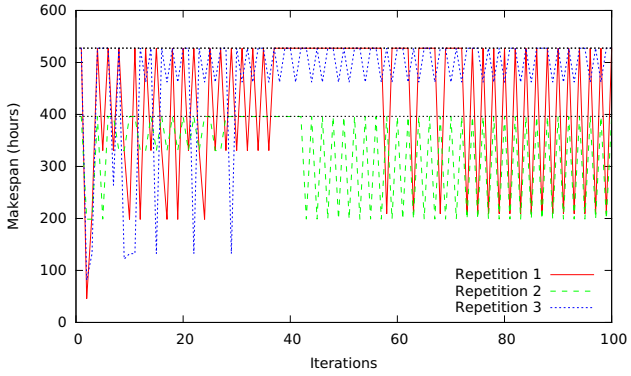
2) Effect of ρ: Figure 3 displays the simulated makespan of our workload executed in configuration **C1** on 5 sites, for $\alpha = 0.5$ and increasing values of ρ. For each repetition, the makespan obtained with the state-of-the-art algorithm is represented with dotted black lines. Repetitions correspond to the same site selections as on Figure 2. ρ mainly regularizes the evolution of the makespan, with little impact on the performance of our algorithm.

3) Effect of the number of sites: Figure 4 displays the simulated makespan of our workload executed in configuration **C1**, for $\alpha = 0.9$, ρ = 0.5, and increasing site numbers. For each repetition, the makespan obtained with the state-of-the-art algorithm is represented with dotted black lines. The behavior is similar for all site numbers: the makespan oscillates between the value obtained on the slowest site selected in the repetition, and lower values reached when the slowest site belongs to a larger class. The slowest site of the infrastructure, which gives a makespan of about 2000 hours, was selected by Repetitions 1 & 3 at 10 sites, by Repetitions 1, 2 & 3 at 20 sites, and obviously at 40 sites. In these cases, our algorithm reaches its full potential, with speed-up factors up to 4. For 40 sites, all repetitions are identical due to the minor influence of ρ.

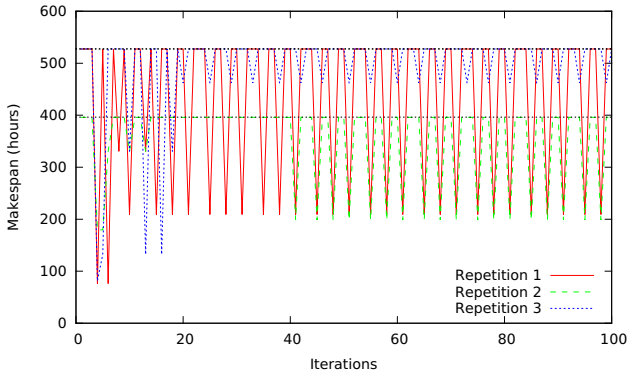
*4) Configurations **C2** and **C3**:* Figure 5 shows the simulation of configurations **C2** and **C3** with 5 sites, for $\alpha = 0.9$ and $\rho = 0.0001$. Repetition 1 in **C3** is identical to Repetition 3 (lines overlap). In **C2** and **C3**, our algorithms do not provide any speed-up compared to the state-of-the-art solution. This is due to the fact that, when only 1 site is used, $\#T_x \cap T_y$ and $\#A_{x,y}$ are always equal to 0, therefore $p = \frac{1}{N'+1}$ which is lower than α.



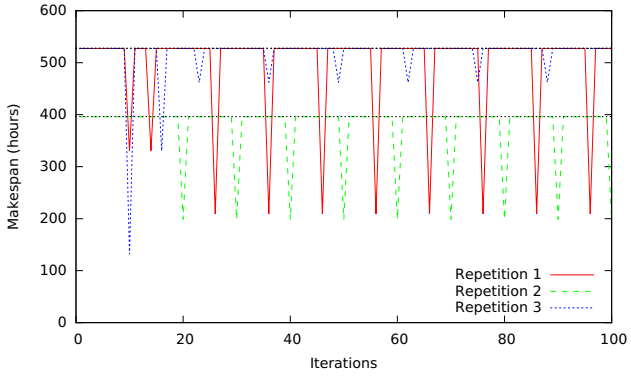
(b) α=0.3



(c) α=0.5

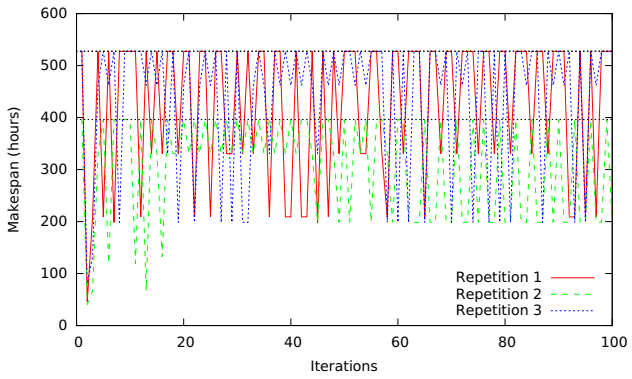


(d) α=0.7

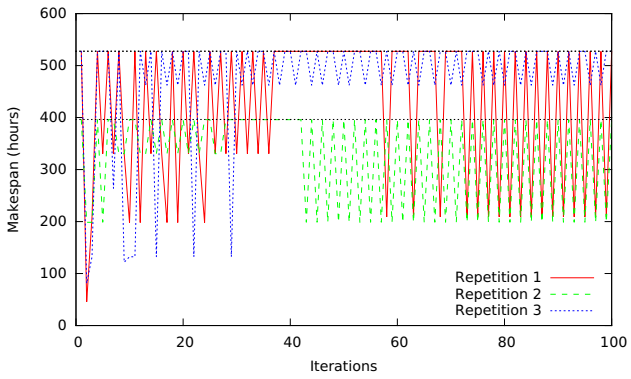


(e) α=0.9

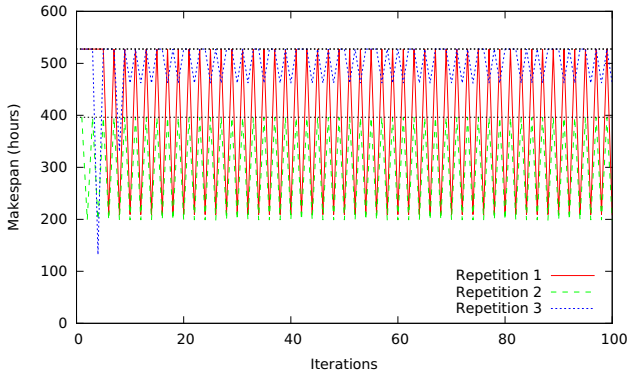
Fig. 2: Effect of α (ρ=0.001, 5 sites, configuration **C1**).



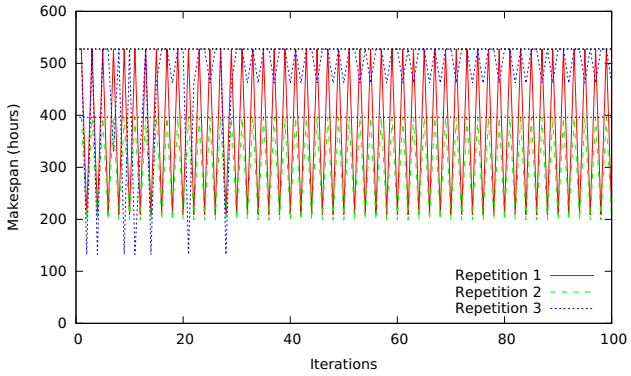
(a) ρ=0.0001



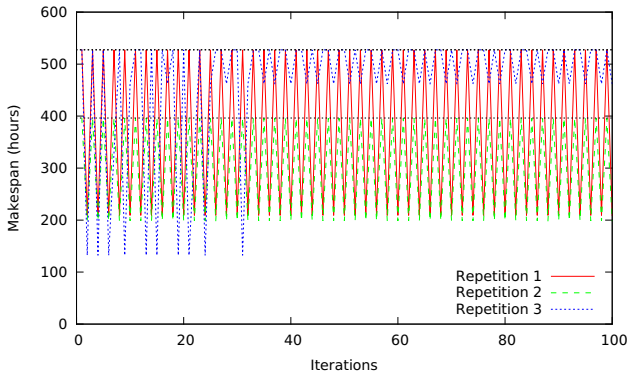
(b) ρ=0.001



(c) ρ=0.01

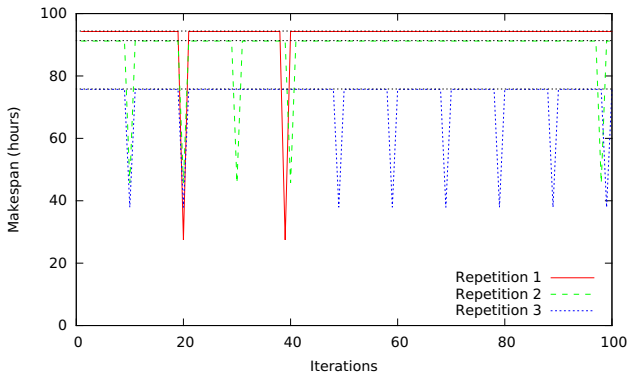


(d) ρ=0.1

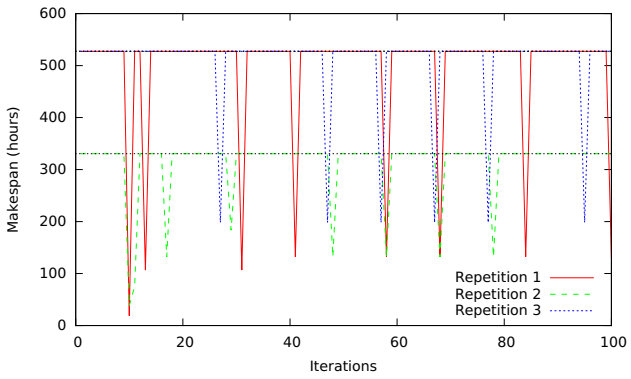


(e) ρ=0.5

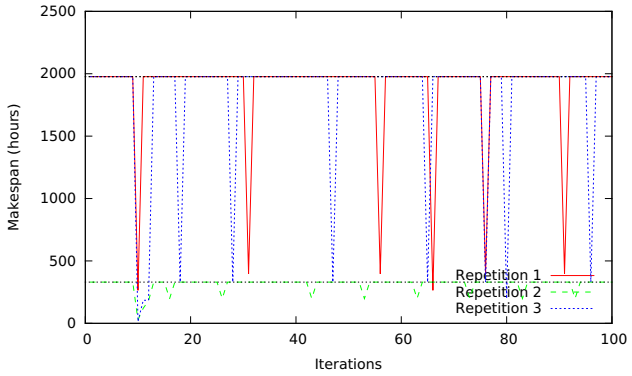
Fig. 3: Effect of ρ (α=0.5, 5 sites, configuration **C1**).



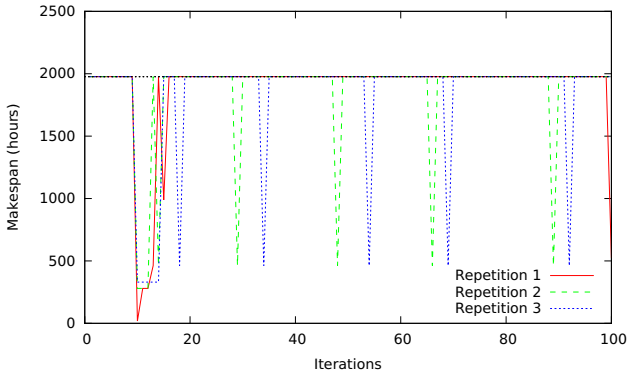
(a) 2 sites



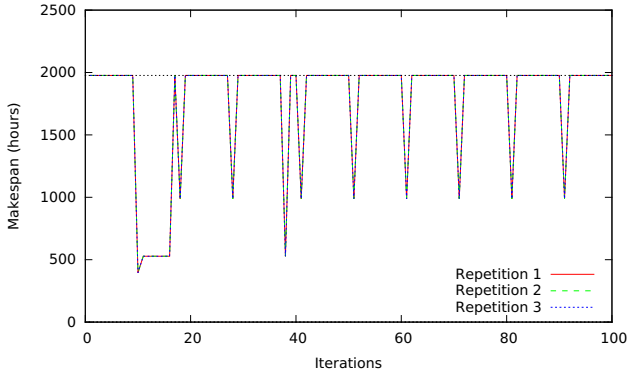
(b) 5 sites



(c) 10 sites

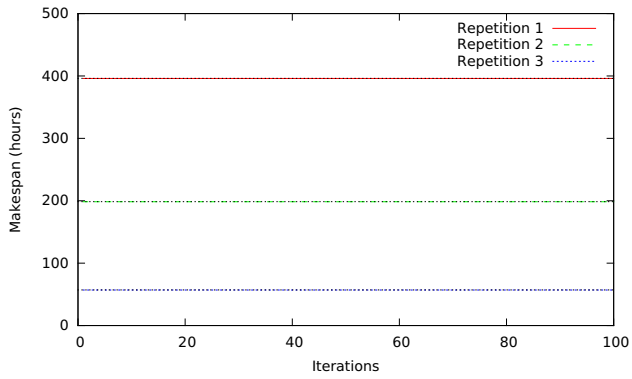


(d) 20 sites

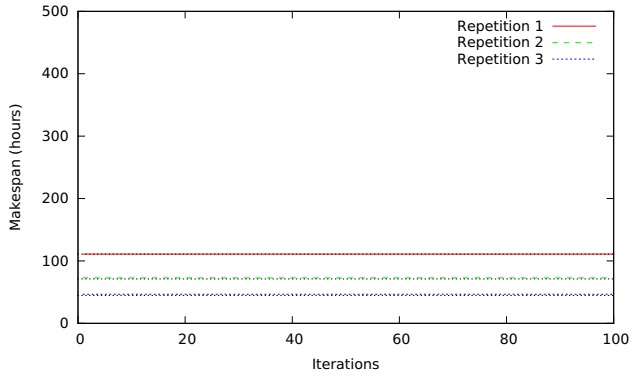


(e) 40 sites

Fig. 4: Effect of site number (α=0.9, ρ=0.0001, configuration **C1**). For readability reasons, graphs for 2 and 5 sites are on a different y scale.



(a) **C2**



(b) **C3**

Fig. 5: Simulation of configurations **C2** and **C3** (α=0.9, ρ=0.0001, 5 sites). For each repetition, the makespan of the state-of-the-art algorithm is represented with dotted black lines.

V. Discussion and Conclusion

We presented a site classification method to address numerical variability among computing sites. Using the theory described in [5], we modeled the risk associated to a classification based on the amount of previously observed numerical accidents. Our model does not make any assumption on the occurrence of numerical accidents, on the workload, or on the infrastructure. In addition, site classifications are continuously updated to integrate new information. Scheduling algorithms addressing **C1**, **C2** and **C3** were derived from the classifications, and evaluated on a simulated grid infrastructure with 40 sites.

In configuration **C1**, we found that the makespan of applications executed with our algorithm oscillates between a maximal value, where a class is created for each computing site, and lower values, where some sites are grouped. Such oscillations are a direct consequence of the need to periodically update the site classification. Our method produces important speed-up factors, up to 11 in our experiments, when high-risk classifications are used (e.g. $\alpha = 0.1$). For lower-risk classifications, a fraction of the executions (about 10% with $\alpha = 0.9$) are significantly accelerated, up to a factor of 4. In the worst case, i.e. when each site has to be assigned to a separate class, our method performs like the state-of-the-art solution. In addition, both the probability α to observe 0 numerical accidents and the probability of accidents ρ regularize the makespan towards a pseudo-periodical function.

The high speed-up factors measured in some parts of our experiment lead us to conclude that site classifications are a promising approach to handle numerical variability among computing sites. Results could be further improved though.

First, our method does not provide any acceleration in configurations **C2** and **C3**, due to the lack of available observations of tasks executed on multiple sites. This could be improved by introducing task replication in these configurations. Accelerations comparable to **C1** would then be expected.

Moreover, in configuration **C1**, *a-priori* information could be introduced to maintain the makespan at the minimum value. For instance, when oscillations are detected, detailed information about the computing system (operating system, versions of dynamic libraries, ...) could be used to aggregate sites *permanently*, or at least until this information changes.

Finally, our classification algorithm could be further investigated. In particular, other algorithms may produce less classes based on the same COMPATIBLE relation.

A few technical obstacles must be addressed before our method can be used in a system deployed in production. First, comparing task results might be difficult when files include context-dependent information such as timestamps, host names, paths and so on. For instance, task results may be compressed archives containing log files. Some data formats may even store provenance information about the process and parameters which produced them, which is for instance the case of the MINC format in medical imaging [10].

When a numerical accident is observed, all previous executions should be reviewed because such new information has a direct impact on the COMPATIBLE relation, therefore on the site classification. A scheduler could address this issue by continuously updating results to increase confidence. For instance, it could first quickly deliver results with a low α value, and then execute additional tasks to increase α up to 1. The fact that our site classification gives an upper bound on the probability to have numerical accidents in the data is useful in this context.

Information about software and hardware upgrades, which is very commonly available to platform administrators, should also be included in the classification. When a site is upgraded, it should be removed from its current class and any information about the tasks executed on this site should be discarded (i.e. T_x should be cleared). Since site classifications are periodically revised, our algorithm would eventually detect such upgrades, but it may be at the cost of several numerical accidents.

Finally, numerical accidents may be inherent in some applications, for instance when pseudo-random numbers are used. To detect such cases, applications should be repeatedly executed on the same system.

VI. Related work

Numerical differences between computing systems have been reported as a potentially serious issue in various fields. For instance, in neuroimaging, the study in [2] reported the effects of Freesurfer version, workstation type, and operating-system version on anatomical volume and cortical thickness measurements. The authors concluded that "users of Freesurfer should exercise caution and restraint before applying a major upgrade in [...] OS version". Based on this study, it is now common practice in this discipline to restrict studies to using homogeneous operating systems and to restrain the use of distributed computing systems. New scheduling methods should patently be developed to address this issue.

Numerical differences between computing sites might occur in any distributed computing system, including cloud when different virtual images are used. Their impact has been studied in the context of desktop grids, where the reliability of resources provided by volunteers has to be assessed. In particular, the BOINC scheduler for desktop grids [11] developed the Homogeneous Redundancy (HR) method that distributes replicas among numerically equivalent machines to recognize erroneous results. As described in [12], this method relies on the following principles:

1) Hosts are assumed numerically equivalent when their operating system and hardware both match.
2) Once the first instance of a task has been sent to a machine, other instance of the same task are sent only to numerically equivalent machines.

The first principle is related to our classification of the computing sites. However, our classification relies only on the observed numerical accidents instead of assuming application models. In practice, it may happen that applications behave identically on two different operating systems (for instance, statically-linked binaries executed on different Linux systems), and that numerical differences are observed when operating system and hardware both match (for instance, applications depending on dynamically-linked libraries). The second principle is related to our scheduling algorithms. Using HR, classes in configuration **C3** would be selected based on the scheduling of only a single task, whereas we ordered the classes by performance values.

Our context is slightly different from desktop grids though. Indeed, in configuration **C1**, we are interested in computing all the possible results on an infrastructure, while desktop grid schedulers usually assume a single correct result, the others being generated by hardware malfunctions, incorrect software modifications or malicious attacks. Therefore, optimization techniques such as presented in [13], where replicated tasks are checkpointed to speed-up the detection of differences, may be only partially applied to our problem.

VII. Acknowledgments

We warmly thank Frédéric Suter for his useful help with the Simbatch simulator. This work is in the scope of the LABEX PRIMES (ANR-11- LABX-0063) of Université de Lyon, within the program "Investissements dAvenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

References

[1] G. A. Lopez, M. Taufer, and P. J. Teller, "Evaluation of IEEE 754 floating-point arithmetic compliance across a wide range of heterogeneous computers," in *Proceedings of the 2007 conference on Diversity in computing*. ACM, 2007, pp. 1–4.

[2] E. H. B. M. Gronenschild, P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, and M. Marcelis, "The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements." *PloS one*, vol. 7, no. 6, p. e38234, Jan. 2012.

[3] T. Sherif, P. Rioux, M.-E. Rousseau, N. Kassis, N. Beck, R. Adalat, S. Das, T. Glatard, and A. Evans, "CBRAIN: A web-based, distributed computing platform for collaborative neuroimaging research," *Frontiers in Neuroinformatics*, vol. 8, no. 54, 2014.

[4] T. Glatard, C. Lartizien, B. Gibaud, R. Ferreira da Silva, G. Forestier, F. Cervenansky, M. Alessandrini, H. Benoit-Cattin, O. Bernard, S. Camarasu-Pop, N. Cerezo, P. Clarysse, A. Gaignard, P. Hugonnard, H. Liebgott, S. Marache, A. Marion, J. Montagnat, J. Tabary, and D. Friboulet, "A Virtual Imaging Platform for multi-modality medical image simulation," *IEEE Transactions on Medical Imaging*, vol. 32, no. 1, pp. 110–118, 2013.

[5] B. Beauzamy, *Méthodes probabilistes pour l'étude des phénomènes réels.*, S. de Calcul Mathématique, Ed., Paris, France, 2014.

[6] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, June 2014.

[7] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, *et al.*, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[8] Y. Caniou and J.-S. Gay, "Simbatch: An API for simulating and predicting the performance of parallel resources managed by batch systems," in *Euro-Par 2008 Workshops-Parallel Processing*. Springer, 2009, pp. 223–234.

[9] B. Fischl, "FreeSurfer." *NeuroImage*, vol. 62, no. 2, pp. 774–81, Aug. 2012.

[10] P. Neelin, D. MacDonald, D. Collins, and A. Evans, "The MINC file format: From bytes to brains," *NeuroImage*, vol. 7, no. 4 PART II, p. S786, 1998, cited By (since 1996)18.

[11] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, 2004, pp. 4–10.

[12] M. Taufer, D. Anderson, P. Cicotti, and C. L. Brooks III, "Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 119a–119a.

[13] F. Araujo, P. Domingues, D. Kondo, and L. M. Silva, "Validating desktop grid results by comparing intermediate checkpoints," in *Achievements in*